Reading to Input Failure, Simple File I/O, Arithmetic Calculations, Using Standard Functions

This programming assignment uses many of the ideas presented in sections 3 and 4 of the course notes, so you are advised to read these notes and the following program specification carefully.

The Program Specification: Newton's Law of Cooling

Suppose an object is heated and then placed in a container filled with some cooler medium (like air, water, oil, etc). Obviously, the temperature of the object will gradually decrease, approaching the temperature of the surrounding medium. For this assignment you will write a simple program that will illustrate this cooling effect by using one of the standard mathematical models that apply to this situation.

Assume that the temperature of the surrounding medium does not change as the object cools. In this case, it is possible to show that the temperature of the object after *t* seconds is modeled by the formula

$$T = \alpha + (T_0 - \alpha) \times e^{-k \times t}$$

where:

- T_0 is the initial temperature of the object (when it is placed in the medium)
- α is the constant temperature of the surrounding medium
- t is the number of seconds since the object was placed in the medium
- k is a constant that is determined by the thermal properties of the object and of the surrounding medium
- e is the Euler number, approximately 2.71828

The rate at which the object is cooling at any instant is also of interest and is given by the formula:

$$R = k \times (T_0 - \alpha) \times e^{-k \times t}$$

For example, suppose that a block of steel is heated to 800° and then placed in a vat of oil at a temperature of 100°. If the constant k equals 0.05 in this case, then after 60 seconds the temperature of the block of steel would be

$$T = 100 + (800 - 100) \times e^{-0.05 \times 60} = 100 + 700 \times e^{-3} \approx 100 + 700 \times 0.0497871 \approx 134.85$$

and the block of steel is cooling at a rate of

$$R = 0.05 \times (800 - 100) \times e^{-0.05 \times 60} = 35 \times e^{-3} \approx 35 \times 0.0497871 \approx 1.74$$

Input file description and sample:

Your program **must** read its input from a file named **`cooling.dat`** — use of another input file name will result in a score of zero. The first line of the input file contains column labels which should be ignored. The second line contains the initial temperature of the object, followed by the constant temperature of the medium. The third line contains a column label which should be ignored. Each remaining line of the input file will contain a single time value in seconds, followed by a single newline character.

You may assume that all the input values will be logically correct (no negatives, for instance). For instance:

```
  Object    Medium
  800.00    100.00
Time
 1.0
 5.0
10.0
20.0
30.0
40.0
50.0
60.0
```

The input values won't always have the nice patterns shown in this example so don't make any unjustified assumptions. Note that you must **also not make any assumptions** about the number of lines of data in the input file. Your program must be written so that it will detect when it's out of input and terminate correctly. A technique for this will be discussed in class; see also slides 4.20 – 4.21 in the course notes.

What to Calculate:

You will write a program to read the input file and use the given values to compute:

- the temperature of the object at the given time
- the rate at which the object is cooling at that time

To perform these calculations, you will need to use the formulas given above. Assume the thermal constant, *k*, is always 0.05. The formula requires using the natural exponential function, to calculate the power of *e*. Fortunately, this may be found among the standard library functions in C++. The natural exponential function is called **`exp( )`**; it takes one parameter of type double and returns the value of *e* raised to that parameter. So, the statement:

```
double aPower = exp(3.14);
```

would assign (approximately) the value 23.1039 to the variable **`aPower`**. To use this function you must include the standard header file **`math.h`**.

Output description and sample:

A sample output file produced from the sample input file above is shown below. The first line of your output should identify you by name, as shown. The second line should include the title "Newton's Law of Cooling" only. The third line should be blank. The fourth and fifth lines should display the initial temperature of the object and of the surrounding medium, labeled exactly as shown.

Next your output file will contain a table, with one line of output for each time value given in the input file. Each line of the table should list the time, the object's temperature at that time, and the rate at which the object is cooling at that time. The table columns should have labels, exactly as shown. There should be a line of delimiters immediately after the column labels, and another to mark the end of the table. Each line of output (including the last) should be followed by a newline character.

The time values should be printed with precision 1, and the temperature and rate values should be printed with precison 2, as shown.

Your program must write its output data to a file named **`temp.out`** — use of any other output file name will result in a score of zero.

```
Programmer:  Donald Allison
Newton's Law of Cooling

Initial temp of object:  800.00
Temperature of medium:   100.00

  Time      Temp      Rate
-------------------------
   1.0    765.86     33.29
   5.0    645.16     27.26
  10.0    524.57     21.23
  20.0    357.52     12.88
  30.0    256.19      7.81
  40.0    194.73      4.74
  50.0    157.46      2.87
  60.0    134.85      1.74
-------------------------
```

You are not required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- You must use variables of type **`double`** for all the real numbers. If you use **`float`** variables, your answers may not be as accurate as needed.
- All decimal values must be printed to show the same number of decimal places as in this sample.
- You must use the specified header and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Use spaces, not tabs to align your output.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line, including the line of hyphens marking the end of the table.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code should be followed. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy I strongly suggest you read the on-line edition (available from the course web page). Some specifics:

- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Use named constants instead of variables where appropriate.
- Precede every major block of your code with a comment explaining its purpose.
- You must use indentation and blank lines to make control structures (like loops and if-else statements) more readable.

Hints:

This program requires that you know how to manage file-oriented input/output operations — the slides and the text provide good examples and guidelines. Your program must read lines of input data until there is no more data to be processed.

You'll have to use **manipulators** to manage the formatting of your output. Read the discussion on slides 4.16 – 4.19 carefully. There are important clues there.

You also have to do some mathematical calculations that go beyond mere arithmetic. The C++ language includes most of the standard mathematical functions, including trigonometric, logarithmic and exponential functions. To use them, you need to add another #include at the beginning of your program: `#include <math.h>`

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen. You should test your program on all the posted input/output examples given along with this specification. Those were generated by the same program that will be used to test your solution.